$1.50

Apple Assembly Line

Volume 2 -- Issue 8 May, 1982

Macro Assembler in EPROM

A number of you have asked about the possiblility of getting the S-C Macro Assembler in EPROM to put on an Apple Firmware card, or a CCS 12K Eprom card. If you want to do it, and know how to modify the firmware card to accept EPROMs, I will send a set of 5 EPROMs containing the assembler for $64. If you also need the monitor in EPROM, add another $16. The assumption will be that you have Applesoft on the mother board, and that you already own the S-C Macro Assembler on disk.

Advertising in AAL

Due to the increased costs of printing more than 1600 copies per month, and with the desire to limit the percentage of advertising pages to less than 30% each month, I have decided to raise the page rate again.

For the June 1982 issue, the price will be $50 for a full page, $30 for a half page. So-called "classified" ads, of up to forty words, will be $5.

The Best Book So Far for Beginners

Roger Wagner's book for beginners wanting to learn assembly language programming is now out, at $19.95. (My price is only $18.) Called "Assembly Lines: The Book", it began as simply a reprint of the series Roger writes for Softalk Magazine. But there is a lot more material in the book, and 100 pages of Appendices. Appendix B, 70 pages, is a very lucid description of every 6502 opcode. If you rank yourself as a beginning assembly language programmer, this book will be a tremendous help.

Secret RWTS Caller Inside DOS.....Bill Parker

I found a portion of code tucked away in DOS that will perform a RWTS for you, doing away with the necessity of finding a place to put a controlling subroutine, an IOB, etc.

As you know, RWTS (Read/Write Track and Sector) gives the programmer the ability to read a sector from any specified track and put it in a buffer in RAM. It also allows the programmer to manipulate the buffer and write it back out to any specified track and sector on the disk.

In this 48K DOS routine, which happens to be the same for 3.3 as well as 3.2, all the programmer has to do is to plug in the track and sector desired and whether he wants to read it from disk to the buffer, or write it from the buffer to the disk. (The buffer is a fixed 256-byte location beginning at $B4BB (46267).) A simple CALL 45111 or a JSR $B037 will then perform the transfer. (You must remember to restore the original Read/Write code back to "2" when you are finished, so that the system can write to the directory when it needs to).

Here is a disassembled and commented version of the routine, which (for lack of a better term) I have named "WRTDIR". This should aid in the development of programs that need to examine or alter the contents of a disk.

This routine, which normally writes a directory sector to the disk from the buffer at $B4BB.B5BA (46267-46522), can be used as a general RWTS utility by plugging in:

Value	Name	$Loc	Loc
Read/Write (1/2)	RW	$B041	45121
Track No. ($0-$22)	TK	$B397	45975
Sector No. ($0-$F)	SC	$B398	45976

Then call 45111 or JSR $B037 and set RW to 2 when done.

```
                     1000          .OR $B037
                     1010          .TF B.WRTDIR
                     1020 *-------------------------------
AAC6-                1030 BUFSTHI  .EQ $AAC6
AAC5-                1040 BUFSTLO  .EQ $AAC5
B052-                1050 CALLRWTS .EQ $B052
B7F0-                1060 IOBBUF   .EQ $B7F0
0002-                1070 RW       .EQ 2
B398-                1080 SC       .EQ $B398
B397-                1090 TK       .EQ $B397
                     1100 *-------------------------------
B037- 20 45 B0       1110 WRTDIR   JSR SETBUFAD
B03A- AE 97 B3       1120          LDX TK
B03D- AC 98 B3       1130          LDY SC
B040- A9 02          1140          LDA #RW
B042- 4C 52 B0       1150          JMP CALLRWTS
                     1160 *-------------------------------
B045- AD C5 AA       1170 SETBUFAD LDA BUFSTLO     PUT BUFFER'S
B048- 8D F0 B7       1180          STA IOBBUF      STARTING ADDRESS IN
B04B- AD C6 AA       1190          LDA BUFSTHI     INPUT OUTPUT BLOCK
B04E- 8D F1 B7       1200          STA IOBBUF+1
B051- 60             1210          RTS
```

Some Patches for the S-C Macro Assembler....Bob Sander-Cederlof

1. Loading the Language Card version: When you type "EXEC LOAD LCASM", the language card is loaded with a copy of the monitor from the Apple mother board, and with the file S-C.ASM.MACRO.LC. The EXEC file also makes a small modification to the memory image depending on which language you have on the mother board.

If you have serial number M-5275 or earlier, the EXEC file does not do the final step of turning on the Assembler. I expected you to type the DOS command "INT" (if Applesoft is on the mother board) or "FP" (if Integer BASIC is on the mother board) to enter the Assembler.

You can make a minor change to the EXEC file to make it automatically turn on the assembler after loading. The next to the last line of the EXEC file is now "C082"; change it to "C080" for automatic turn-on.

Here is a step-by-step procedure for the change. Try it on a COPY of the master disk, in case you make a mistake.

1. Get into the S-C Macro Assembler, either regular of language card version, it doesn't matter which).

2. Type "AUTO". The Assembler will print ":1000 " and wait for input.

3. Type five (5) backspaces (left arrow) to position the cursor right after the prompt.

4. Type "EXEC LOAD LCASM", and the Assembler will load the EXEC file into memory. You will see a list of line numbers on the screen.

5. Type five backspaces and the word "MANUAL" to turn off the auto-line-number mode.

6. LIST the lines (type "LIST").

7. See line 1090? It should be "C082". Type "1090 C080" to change it.

8. Type "TEXT LOAD LCASM" to save the modified version.

9. That's all there is to it!

By the way, Bob Potts (from the Bank Of Louisville) was here last week. He brought along a Corvus 5-meg drive, so we put the Language Card version of the Assembler on it. For some reason which we can't explain, the EXEC file hangs up after the BLOAD (but only if the language card has not been loaded since power up). We changed the EXEC file slightly, and it always worked. Instead of doing the BLOAD while in the monitor, we did it from Applesoft. Here is the new version of the file:

```
REM LOAD S-C MACRO ASSEMBLER
REM INTO THE LANGUAGE CARD
CALL-151
C081 C081
F800<F800.FFFFM
BLOAD S-C.ASM.MACRO.LC
300:A9 4C CD 00 E0 F0 12 8D 00 E0 A9 00 8D 01 E0 A9 D0 8D
02 E0 A9 CB 8D D1 03 60
300G
C080
3D3G
```

You may also want to change the HELLO file to include an option to EXEC LOAD LCASM automatically.

2. If you have a Language Card, and your Assembler has a serial number of about M-5030 or earlier, the memory limits are not set up properly in all cases.

Check your copy of S-C.ASM.MACRO.LC by loading it into the language card and typing "$D2C6" from inside the assembler. If you don't have $A0 there, then you need to install this patch:

1. Type in these monitor commands:

```
:$D2C6:A0 0C
:$D2C8:20 1E D3 A9 00 91 58 85
:$D2D0:D9 AD 00 E0 C9 4C F0 08
```

2. Type "BSAVE S-C.ASM.MACRO.LC,A$D000,L$231F".

3. If you have serial number M-5287 or earlier, a more difficult to apply patch is needed to correct a problem in printing the symbol table. If you are using the .TI directive, and if you have several lines of local labels in the symbol table listing, and if the page break comes between two such lines, the listing is messed up in a disastrous way.

To fix the S-C.ASM.MACRO, do the following (very carefully):

1. Get into the assembler by typing "BRUN S-C.ASM.MACRO".

2. Type the following monitor commands:

```
:$2AF<26AF.26D5M
:$26B1<2AF.2D5M
:$26AF:84 2F
:$26C1:CA
```

3. Type "BSAVE S-C.ASM.MACRO,A$1000,L$21D3"

To repair the language card version, do the following:

1. EXEC LOAD LCASM, and get into the assembler by typing INT (unless you already made the change to the EXEC file noted above).

2. Type the following monitor commands:

```
:$C083 C083
:$2AF<E7FB.E821M
:$E7FD<2AF.2D5M
:$E7FB:84 2F
:$E80D:16
```

3. Type "BSAVE S-C.ASM.MACRO.LC,A$D000,L$231F"

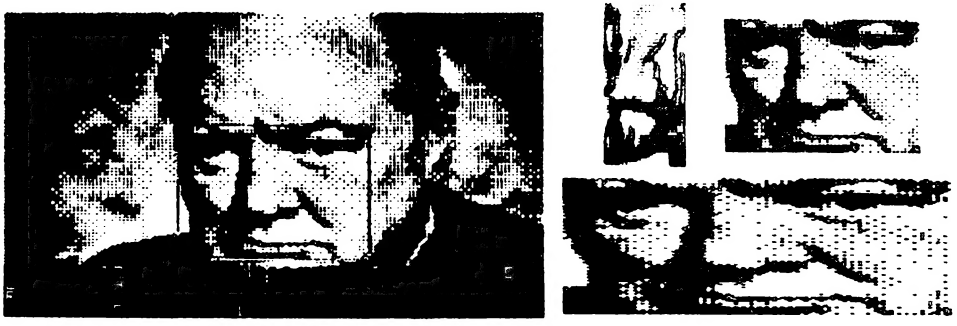
If these patches and my instructions seem too difficult, you can send me $2.50 and your S-C Macro Assembler diskette; I will update it with the new HELLO program, the new LOAD LCASM file, and the patched copies of the assembler.

Vinyl Diskette Pages for your S-C Assembler Binder

I am having 1000 special pages manufactured. They will fit the binder that comes with the Macro Assembler, and will hold one diskette each and a 3x5 index card. For $6 I'll send you ten of them. For $12 I'll send them in a binder. For $36 you can have a binder with ten blank diskettes in vinyl pages. The binder is also just right for storing back issues of AAL.

EPSON HIEROGRAPHIC TRANSPORT

PRINTER GRAPHICS SOFTWARE



TRANSPORT FEATURES: WINDOW CONTROL ON HI-RES SCREEN (AS SHOWN ABOVE) ALLOWING ANY RECTANGULAR AREA OF THE SCREEN TO BE SELECTED FOR PRINT; VARIABLE SCALE IN BOTH HEIGHT AND WIDTH OF THE PICTURE ALLOWING STRETCH AND MAGNIFICATION; NORMAL OR ROTATED PICTURE; POSITIVE OR NEGATIVE PRINT; PICTURE FILES LOADED WITH A SINGLE KEY PRESS; MODULE INCLUDED THAT MAY BE CALLED FROM BASIC; WRITTEN IN MACHINE LANGUAGE.

FOR 48K APPLE II/APPLE II PLUS WITH EPSON MX-70/80/100 PRINTER (MX-80 MUST HAVE GRAFTRAX 80 GRAPHICS OPTION)

APPLE II / APPLE II PLUS ARE TRADEMARKS OF APPLE COMPUTER, INC.
EPSON MX-70/80/100 AND GRAFTRAX 80 GRAPHICS ARE TRADEMARKS OF EPSON AMERICA, INC.

THE CLONE KIT

NOW $34.95

CLONE KIT IS A MULTI-PURPOSE BIT COPIER FOR THE APPLE II COMPUTER, GIVING YOU THE MEANS TO PROTECT YOUR INVESTMENT IN SOFTWARE BY BACKING UP MOST OF YOUR "PROTECTED" DISKETTES.

WITH CLONE KIT YOU CAN:

- BACKUP DISKETTES USING SYNCHRONIZED TRACKS AS PROTECTION
- BACKUP DISKETTES USING A MODIFIED DOS
- COPY ONLY SELECTED TRACKS (USEFUL TO RESTORE A "BLOWN" DOS)
- MAKE A SINGLE DRIVE BACKUP
- DUPLICATE DISKS USING SPECIAL TIMING FOR PROTECTION
- PROVIDES EXTENDED INFORMATION FOR VIEWING THE DISKETTE STRUCTURE
- COPY PASCAL AND CP/M DISKETTES QUICKLY AND EASILY

WORKS WITH BOTH 13 AND 16 SECTOR DISKETTES

WRITE OR CALL (214) 259-6482 OR (214) 259-5196

HIEROGRAPHIC TRANSPORT ... $39.95
THE CLONE KIT ..... ~~$49.95~~ $34.95

GSR ASSOCIATES - P.O. BOX 401462 - GARLAND, TEXAS 75040

DEALER INQUIRIES INVITED

Checks, M.O.

Benchmarking Block MOVES.....William R. Savoie

While working on my new soon-to-be-released data file system, I came to see the importance of a speedy 6502 block move routine. I have resisted "moves" like the coming of winter. I have a super two pass sort routine that is very fast. Providing a person has less than 2000 files there is no need to do much file moving, since only pointers need move. (My system has a 64K card, giving me a 112K system.)

Unfortunately, the real world needs some 10,000 or more files, and these of course must be sorted too. By physically moving the files as directed by the sorted pointers, and then moving all this to disk, it is possible to use a merge sort to get the whole job done in the least amount of time. With this preamble behind us, let's get on the move!

I have benchmarked three approaches to moving blocks: the monitor move down (located at $FE2C) which I'm sure you all have used, and its variation, a similar move up routine. Next is the Applesoft block move, and third is a self modifying move which I call "Quick Move".

To ease such a tedious undertaking, I have included a BASIC connection to pass variables and determine the benchmark precision. To help further, I have added a hex converter, a memory dump routine, and an automatic 3D0G vector using the ctrl-Y command from the monitor. To help with the problem of what block of memory was moved where, I wrote a memory fill routine. This acts to place the memory address back into memory, on two-byte boundaries. You can easily read memory to see where it came from.

My first benchmark was a block move of 10,000 bytes made 100 times. The next was a move of 10.240 bytes, again made 100 times. Here are the conditions and resulting times:

		mon up	mon dn	AS	QM
Case I	Lo=18674=$48F0	47	53	17.2	15.3
seconds					
Case II	Lo=18432=$4800	48.7	54.7	16.7	14.7

Note: 0.5 seconds of these values due to BASIC overhead.

As you can see, the old monitor move is not made for high speed moves. For one thing, a two-byte subtraction is carried out for each byte that is moved. It is much more efficient to do the subtraction only once, before you start. A closer look shows that it is faster to move more data, providing you move a whole number of memory pages! The time needed to move the "extra" 240 bytes was negative 0.5 seconds for the Applesoft block move and negative 0.6 seconds for the "Quick Move". There was no sensitivity to start and destination boundaries. "Quick Move" was 3.7 times faster than the monitor move!

I tried putting the first half of Quick Move on page zero at $A0, but the speed improvement was only 0.7 seconds (about 5%)

over the time it took when located at $3000.

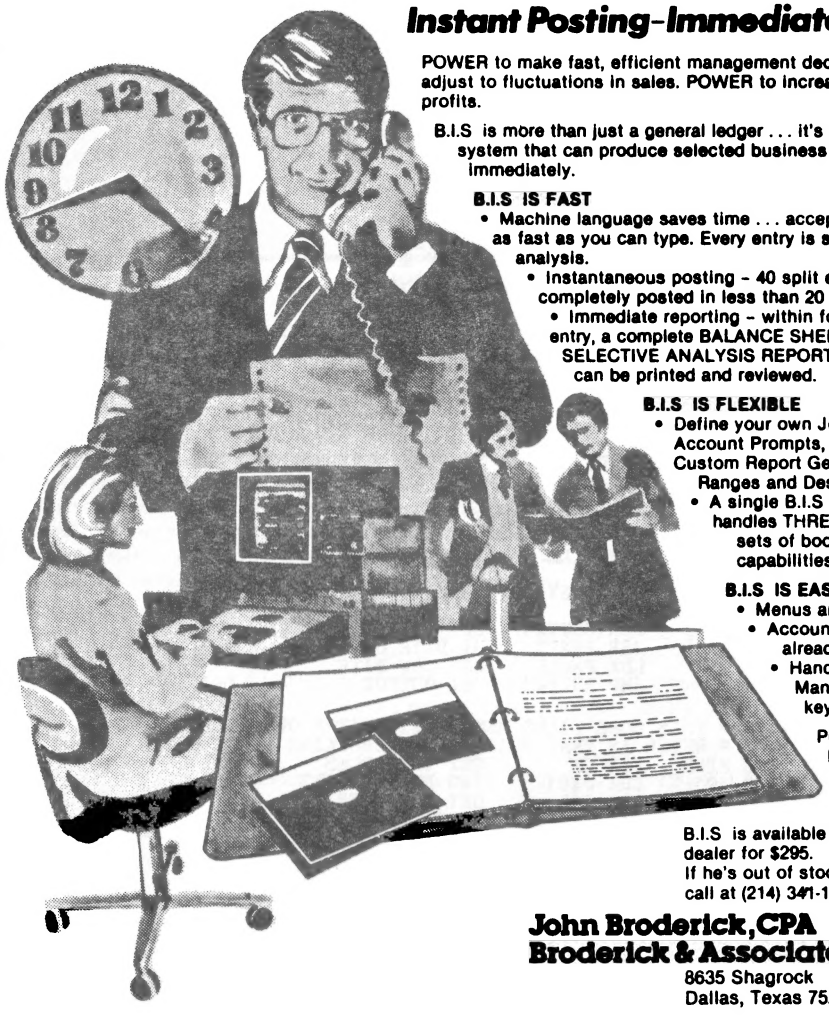
As a further note, each move routine requires its own parameter organization. If files are to be moved and not lost, attention must be paid to exact specification of end points and lengths.

```
100  GOSUB 300: REM INITIALIZE BINARY PROGRAM
105 A = 0:B = 0:C = 0:J = 0
106  TEXT : HOME : PRINT : PRINT " BLOCK MOVE TESTING"; SPC( 8)"L
     OOP = ";L + 1: PRINT : PRINT "(0) END" SPC( 13)"(1) TEST OVE
     RHEAD": PRINT : PRINT "(2) M.MOVE UP" SPC( 7)"(3) M.MOVE DOW
     N"
107  PRINT : PRINT "(4) A.MOVE UP" SPC( 7)"(5) QUICK MOVE DOWN": PRI
     : PRINT "(6) HEX DUMP" SPC( 8)"(7) ADDRESS FILL"
108  PRINT : PRINT "(8) HEX CONVERT" SPC( 5)"(9) EDIT": PRINT : PRIN
     "(10) MONITOR WITH CTRL-Y RETURN": PRINT
109  PRINT "(11) SET CALL LOOP = 100": PRINT
110  PRINT "PICK TEST NUMBER (0 TO 11) ";: INPUT I: IF I < 0 OR I
      > 11 THEN 110
112 HI = 28672: REM $7000 HI BLOCK
114 LO = 18672: REM $48F0 LO BLOCK
115 LO = 18432: REM $4800 LO BLOCK
118 LE = 2056: REM $808 LENGTH OF MOVE
130  ON I GOSUB 150,160,170,180.190,200,210,220,230,240,250
134  IF I = 0 THEN  END
136  IF I > 5 THEN  PRINT : PRINT "HIT ANY KEY ";: GET A$: GOTO 1
     05
140  PRINT  CHR$ (7);I;" DONE ";J;" TIMES": GET A$: GOTO 105
150 A = LO:B = LO:C = LO + 1: FOR J = 0 TO L: CALL BASE,A,B,C: NEXT
     : RETURN : REM TIME BASIC
160 A = LO:B = HI:C = HI + LE: FOR J = 0 TO L: CALL BASE,A,B,C: NEXT
     : RETURN : REM M.MOVE UP
170 A = LO:B = HI:C = LO - LE - 1: FOR J = 0 TO L: CALL BASE,A,B,
     C: NEXT : RETURN : REM M.MOVE DOWN
180 A = HI:B = LO:C = HI + LE: FOR J = 0 TO L: CALL BASE + 3,A,B,
     C: NEXT : RETURN : REM A.MOVE UP
190 A = LO:B = LO - LE:C = HI - LO: FOR J = 0 TO L: CALL BASE + 6
     ,A,B,C: NEXT : RETURN : REM QUICK MOVE DOWN
200  CALL BASE + 9,LO - LE - 32,0,HI + LE + 32: RETURN
210  CALL BASE + 12,LO - LE - 16.HI + LE + 16,0: RETURN
220  HOME : VTAB 8: PRINT "INPUT DECIMAL NUMBER ";: INPUT N: CALL
     BASE + 15,N: RETURN : REM HEX CONVERTER
230  TEXT : HOME : POKE 33,33: LIST : END
240  HOME : FOR I = 1017 TO 1018:A =  PEEK (I - 40): POKE I,A: NEXT
     : POKE 1016,76: POKE 72,0: CALL  - 151
250 L = 99: GOTO 105
300  HIMEM: 12287: REM $2FFF
305 BASE = 12288: IF  PEEK (BASE) = 76 THEN  RETURN : REM .OR $30
     00
310 D$ =  CHR$ (13) +  CHR$ (4)
320  PRINT D$"BLOAD B.BLOCK MOVE BENCHMARKS"
330  PRINT : RETURN
```

B.I.S

Instant Information is Power

GENERAL LEDGER ON LINE



Instant Posting-Immediate Reporting

POWER to make fast, efficient management decisions. POWER to adjust to fluctuations in sales. POWER to increase productivity and profits.

B.I.S is more than just a general ledger . . . it's business information system that can produce selected business activity reports immediately.

B.I.S IS FAST

- Machine language saves time . . . accepts 3000 entries a day as fast as you can type. Every entry is saved for selective analysis.
- Instantaneous posting - 40 split entries can be completely posted in less than 20 seconds.
- Immediate reporting - within four minutes of any entry, a complete BALANCE SHEET, P&L, or DETAILED SELECTIVE ANALYSIS REPORT reflecting that entry can be printed and reviewed.

B.I.S IS FLEXIBLE

- Define your own Journal Names, Account Prompts, Activity Codes, Custom Report Generators, Account Ranges and Descriptions.
- A single B.I.S General Ledger handles THREE completely separate sets of books with consolidating capabilities.

B.I.S IS EASY TO USE

- Menus are simple.
- Accounts and Journals are already set up.
- Hand-holding Tutorial Manual outlines every key stroke.

Put the power of B.I.S to work for you.

B.I.S is available from your local Apple dealer for $295.
If he's out of stock, have him give us a call at (214) 341-1635 or write:

John Broderick, CPA
Broderick & Associates
8635 Shagrock
Dallas, Texas 75238

```
                    1000 ********************************
                    1010 *                              *
                    1020 *  BENCHMARKING BLOCK MOVES     *
                    1030 *                              *
                    1040 *  BY WILLIAM R. SAVOIE  3/82  *
                    1050 *   LIMERICK TRAINING CENTER    *
                    1060 *  C/O GENERAL PHYSICS CORP.   *
                    1070 *  341 LONGVIEW RD., LINFIELD  *
                    1080 *    PENNSYLVANIA   ZIP 19468  *
                    1090 ********************************
                    1100
                    1110
                    1120 *--------------------------------*
                    1130 * APPLE II PAGE ZERO MEMORY USE *
                    1140 *--------------------------------*
                    1150
003C-               1160 A1L     .EQ $3C       MONITOR
003D-               1170 A1H     .EQ $3D        USE
003E-               1180 A2L     .EQ $3E        FOR
003F-               1190 A2H     .EQ $3F        BLOCK
0040-               1200 A3L     .EQ $40        MOVE
0041-               1210 A3H     .EQ $41
0042-               1220 A4L     .EQ $42
0043-               1230 A4H     .EQ $43
00A0-               1240 FACMO   .EQ $A0       FP REGISTER
00A1-               1250 FACLO   .EQ $A1       FP REGISTER
                    1260
                    1270 *--------------------------------*
                    1280 * OTHER APPLE II MEMORY MAPPING *
                    1290 *--------------------------------*
                    1300
D39B-               1310 BLTU    .EQ $D39B     BLOCK TRANSFER
DD67-               1320 FRMNUM  .EQ $DD67     FORMULA=>NUM
DEBE-               1330 COMA    .EQ $DEBE     CHECK COMA
E10C-               1340 AYINT   .EQ $E10C     MAKE INTEGER
F944-               1350 PRNTX   .EQ $F944     PRINT X
FCBA-               1360 NXTA1   .EQ $FCBA     INCR POINTER
FDDA-               1370 PRBYTE  .EQ $FDDA     PRINT A
FE2C-               1380 MOVE    .EQ $FE2C     MONITOR MOVE
                    1390
                    1400
                    1410 *--------------------------------*
                    1430      .OR $3000
                    1440      .TF B.BLOCK MOVE BENCHMARKS
                    1450 *--------------------------------*
                    1460
                    1470 *  THIS CODE ALLOWS SIMPLE ENTRY WITHOUT THE & COMMAND
3000- 4C 7D 30      1480 BEGIN   JMP MONITOR.MOVE
3003- 4C B0 30      1490         JMP APPLESOFT.MOVE
3006- 4C CE 30      1500         JMP QUICK.MOVE
3009- 4C 31 30      1510         JMP DUMP      HEX OUTPUT
300C- 4C 1A 30      1520         JMP FILL      LABLE MEMORY
                    1530
                    1540 * TO HELP A HEX CONVERTER
                    1550 CONVERT
300F- 20 59 30      1560         JSR GETVAR    GET VARIABLE
3012- 20 44 F9      1570         JSR PRNTX     HI BYTE OUT
3015- A5 A1         1580         LDA FACLO     GET LOW BYTE
3017- 4C DA FD      1590         JMP PRBYTE    HEX OUTPUT

                    1610 * THIS CODE FILLS MEMORY WITH IT'S OWN ADDRESS
                    1620 * WHICH IS VERY USEFULL FOR CHECKING BLOCK MOVES
301A- 20 67 30      1630 FILL    JSR GM        GET VARIABLES
301D- A0 01         1640 .01     LDY #$01      TWO BYTE OFFSET
301F- A5 3C         1650         LDA A1L       GET LOW BYTE
3021- 91 3C         1660         STA (A1L),Y   WRITE ADDRESS LO
3023- 88            1670         DEY           MOVE LEFT
3024- A5 3D         1680         LDA A1H       GET HI ADDRES
3026- 91 3C         1690         STA (A1L),Y   WRITE TO MEMORY
3028- 20 BA FC      1700         JSR NXTA1     INCREMENT PTR
302B- 20 BA FC      1710         JSR NXTA1     TWICE
302E- 90 ED         1720         BCC .01       GO TELL DONE
3030- 60            1730         RTS
```

```
                        1740
                        1750 * A UTILITY DUMP TO SEE MEMORY FROM BASIC
3031- 20 67 30          1760 DUMP   JSR GM        GET VARS
3034- AD 10 C0          1770        LDA $C010     CLEAR STROBE
3037- AD 00 C0          1780 .01    LDA $C000     GET KEY IF ONE
303A- 10 0F             1790        BPL .03       GO DUMP HEX
303C- AD 10 C0          1800        LDA $C010     CLEAR STROBE
303F- AD 00 C0          1810 .02    LDA $C000     READ KEY AGAIN
3042- 10 FB             1820        BPL .02       WAIT FOR KEY
3044- C9 8D             1830        CMP #$8D      'RETURN' KEY?
3046- F0 10             1840        BEQ .04       EXIT
3048- AD 10 C0          1850        LDA $C010     CLEAR STROBE
304B- 20 A3 FD          1860 .03    JSR $FDA3     8 HEX OUT
304E- A5 3E             1870        LDA A2L       END LOW
3050- C5 A1             1880        CMP $A1       DESIRED LOW
3052- A5 3F             1890        LDA A2H       HI TOO
3054- E5 A0             1900        SBC $A0       ENOUGH?
3056- 90 DF             1910        BCC .01       GO TELL DONE
3058- 60                1920 .04    RTS
                        1930
                        1940 * GET VARIABLE FROM BASIC
                        1950 * MUST BE <65357 OR SYNTAX ERR
                        1960 * PLACE IN REGISTERS X AND A
                        1970
3059- 20 BE DE          1980 GETVAR JSR COMA      MUST SEE COMA
305C- 20 67 DD          1990        JSR FRMNUM    GET NUMBER
305F- 20 0C E1          2000        JSR AYINT     MAKE INTEGER
3062- A6 A0             2010        LDX FACMO     HI BYTE
3064- A5 A1             2020        LDA FACLO     LOW BYTE
3066- 60                2030        RTS
                        2040
```

```
                        2050
                        2060 * GET BASIC VARIABLES INTO THE
                        2070 * MONITORS WORK REGISTERS USED BY
                        2080 * COMMANDS M,V,G,L,S,T,-,+,..ETC
                        2090
3067- 20 59 30          2100 GM      JSR GETVAR    BLOCK START
306A- 85 3C             2110         STA A1L       LOW BYTE
306C- 86 3D             2120         STX A1H       HI BYTE
306E- 20 59 30          2130         JSR GETVAR    BLOCK END
3071- 85 3E             2140         STA A2L       LOW
3073- 86 3F             2150         STX A2H       HI BYTE
3075- 20 59 30          2160         JSR GETVAR    DEST. START
3078- 85 42             2170         STA A4L       LOW
307A- 86 43             2180         STX A4H       HI
307C- 60                2190         RTS

                        2210 *-------------------------------*
                        2220 * THE OLD MONITOR MOVE           *
                        2230 * MOVE BLOCK OF MEMORY UP/DOWN   *
                        2240 *-------------------------------*
                        2250
                        2260 MONITOR.MOVE
307D- 20 67 30          2270         JSR GM        SET UP MOVE
3080- A5 3C             2280         LDA A1L       START LOW
3082- C5 42             2290         CMP A4L       END LOW
3084- A5 3D             2300         LDA A1H       START HI
3086- E5 43             2310         SBC A4H       WHICH BIGGER?
3088- B0 21             2320         BCS MOVEDN    GO DOWN IN MEM
                        2330
308A- A0 00             2340 MOVEUP  LDY #$00      CLEAR INDEX
308C- B1 3E             2350 .01     LDA (A2L),Y     GET DATA
308E- 91 42             2360         STA (A4L),Y     PUT DATA
3090- A5 42             2370         LDA A4L       GET INDEX
3092- D0 02             2380         BNE .02       PAGE CROSS?
3094- C6 43             2390         DEC A4H       HI BYTE
3096- C6 42             2400 .02     DEC A4L       LOW BYTE
3098- A5 3E             2410         LDA A2L
309A- C5 3C             2420         CMP A1L       END YET?
309C- A5 3F             2430         LDA A2H
309E- E5 3D             2440         SBC A1H
30A0- A5 3E             2450         LDA A2L
30A2- D0 02             2460         BNE .03       PAGE CROSS?
30A4- C6 3F             2470         DEC A2H       HI BYTE
30A6- C6 3E             2480 .03     DEC A2L       LOW BYTE
30A8- B0 E2             2490         BCS .01       GO TELL DONE
30AA- 60                2500         RTS
                        2510
30AB- A0 00             2520 MOVEDN  LDY #$00      CLEAR INDEX
30AD- 4C 2C FE          2530         JMP MOVE      MONITOR MOVE
                        2540
                        2550
                        2560 *-------------------------------*
                        2570 * AND ALONG CAME THE PEOPLE AT *
                        2580 * MICROSOFT WITH THEIR MOVE    *
                        2590 *-------------------------------*
                        2600
                        2610 APPLESOFT.MOVE
30B0- 20 59 30          2620         JSR GETVAR    HI ADDRESS OF BLOCK TO MOVE
30B3- 85 96             2630         STA $96       LOW
30B5- 86 97             2640         STX $97       HI BYTE
30B7- 20 59 30          2650         JSR GETVAR    BLOCK END
30BA- 48                2660         PHA           SAVE TELL LAST
30BB- 8A                2670         TXA           NEED 9B,9C
30BC- 48                2680         PHA           TO GETVARS
30BD- 20 59 30          2690         JSR GETVAR    HI ADDRESS OF DISTINATION
30C0- 85 94             2700         STA $94       LO&HI BYTES
30C2- 86 95             2710         STX $95
                        2720
                        2730 * WE USED $9B,9C TO GET THE TWO BYTE FP VALUE
30C4- 68                2740         PLA           END OF BLOCK
30C5- 85 9C             2750         STA $9C       HI BYTE
30C7- 68                2760         PLA
30C8- 85 9B             2770         STA $9B       LOW BYTE TOO
30CA- 38                2780         SEC           SUBTRACT COMMING
30CB- 4C 9B D3          2790         JMP BLTU      A.MOVE
```

```
                        2810 *--------------------*
                        2820 * MOVING IN RAM CAN *
                        2830 * BE EVEN FASTER    *
                        2840 *--------------------*
                        2850
                        2860 QUICK.MOVE
30CE- 20 59 30          2870         JSR GETVAR    GET START
30D1- 8D EF 30          2880         STA .01+1     LOW BYTE
30D4- 8D 11 31          2890         STA .06+1     COPY HERE TOO
30D7- 8E F0 30          2900         STX .01+2     HI
30DA- 20 59 30          2910         JSR GETVAR    DESTINATION
30DD- 8D F2 30          2920         STA .02+1     LO
30E0- 8D 14 31          2930         STA .07+1     COPY HERE TOO
30E3- 8E F3 30          2940         STX .02+2     HI BYTE
30E6- 20 59 30          2950         JSR GETVAR    END ADDRESS
30E9- 8A                2960         TXA           SET TEST FOR
30EA- F0 14             2970         BEQ .05        MOVE<256?
                        2980
                        2990 * X=PAGE NUMBERS TO MOVE
30EC- A0 00             3000         LDY #$00      INDEX=0
30EE- B9 00 48          3010 .01     LDA $4800,Y   SOURCE
30F1- 99 00 40          3020 .02     STA $4000,Y   DESTINATION
30F4- C8                3030         INY           NEXT BYTE
30F5- D0 F7             3040         BNE .01       SMALL MOVE
30F7- EE F0 30          3050 .03     INC .01+2     HI SOURCE
30FA- EE F3 30          3060 .04     INC .02+2     HI DESTINATION
30FD- CA                3070         DEX           DONE?
30FE- D0 EE             3080         BNE .01       Y=0 SO MOVE PAGE
                        3090
                        3100 * SET UP REMAINING MOVE
3100- A4 A1             3110 .05     LDY $A1       LOW BYTE LENGTH
3102- F0 16             3120         BEQ .08       GO IF NONE
3104- AD F0 30          3130         LDA .01+2     COPY HI BYTE
3107- 8D 12 31          3140         STA .06+2     FOR SOURCE
310A- AD F3 30          3150         LDA .02+2      AND
310D- 8D 15 31          3160         STA .07+2      DESTINATION
                        3170
                        3180 * NOW WITH X=0 START MOVING LOW BYTE OF LENGTH
                        3185 * (Y) = REMAINING BYTES TO MOVE
3110- BD 00 48          3190 .06     LDA $4800,X   SOURCE
3113- 9D 00 40          3200 .07     STA $4000,X   DESTINATION
3116- E8                3210         INX           NEXT
3117- 88                3220         DEY           MOVE ENOUGH?
3118- D0 F6             3230         BNE .06       GO TELL DONE
311A- 60                3240 .08     RTS
```

New AED Features.....Bob Sander-Cederlof

Bill Linn, author of AED, stopped by the other day. Bill is a Vice President at Cullinane Corporation, and was in Dallas for a user convention. Since it was Sunday, and we are both earnest Christians, he spent the morning with Becky and me and the kids at church. Later we all went out for an excellent lunch at the local Harvey House.

He brought the latest version of AED along, and showed me all the new features. AED now has keyboard macros! They are user definable, but he has predefined quite a few for you. If you type two escapes in a row, the top 18 lines of the screen are used to display a menu of all the currently defined escape macros. Escape followed by some other character inserts the corresponding text string at the current cursor position.

There is a utility program on the disk for use in defining your own macro strings, and it is very easy to use. In fact, you use AED editing to modify simple DATA statements within the utility itself. When you are through with your changes, the utility modifies the macro table within AED and on the disk.

Again I say, if you are not fully satisfied with your current stable of Applesoft programming aids, you owe it to yourself to buy AED. It will save you countless hours of frustrating retyping as you create and edit and restructure and debug and modify Applesoft programs.

Decision Systems

Decision Systems
P.O. Box 13006
Denton, TX 76203
817/382-6353

DIS-ASSEMBLER

DSA-DS dis-assembles Apple machine language programs into forms compatible with LISA, S-C ASSEMBLER (3.2 or 4.0), Apple's TOOL-KIT ASSEMBLER and others. DSA-DS dis-assembles instructions or data. Labels are generated for referenced locations within the machine language program.
$25, Disk, Applesoft (32K, ROM or Language card)

OTHER PRODUCTS

ISAM-DS is an integrated set of Applesoft routines that gives indexed file capabilities to your **BASIC** programs. Retrieve by key, partial key or sequentially. Space from deleted records is automatically reused. Capabilities and performance that match products costing twice as much.
$50 Disk, Applesoft.

PBASIC-DS is a sophisticated preprocessor for structured **BASIC**. Use advanced logic constructs such as **IF...ELSE...**, **CASE**, **SELECT**, and many more. Develop programs for Integer or Applesoft. Enjoy the power of structured logic at a fraction of the cost of **PASCAL**.
$35. Disk, Applesoft (48K, ROM or Language Card).

FORM-DS is a complete system for the definition of input and output froms. **FORM-DS** supplies the automatic checking of numeric input for acceptable range of values, automatic formatting of numeric output, and many more features.
$25 Disk, Applesoft (32K, ROM or Language Card).

UTIL-DS is a set of routines for use with Applesoft to format numeric output, selectively clear variables (Applesoft's **CLEAR** gets everything), improve error handling, and interface machine language with Applesoft programs. Includes a special load routine for placing machine language routines underneath Applesoft programs.
$25 Disk, Applesoft.

SPEED-DS is a routine to modify the statement linkage in an Applesoft program to speed its execution. Improvements of 5-20% are common. As a bonus, **SPEED-DS** includes machine language routines to speed string handling and reduce the need for garbage clean-up. Author: Lee Meador.
$15 Disk, Applesoft (32K, ROM or Language Card).

(Add $4.00 for Foreign Mail)

***Apple II is a registered trademark of the Apple Computer Co.**

Another Recursive Macro.....Lee Meador

Last month I sent Bob a recursive macro definition that he put in the AAL for everyone to see. In case you have forgotten what recursive means, let me explain it somewhat. If you have a macro thatcalls itself under certain conditions, that macro is called 'recursive'. It's kind of like the plastic cup I had when I was little. There was a picture on the cup of a little bear sitting in a high-chair. The bear was holding a plastic cup and on the cup was a picture of a little bear in a high-chair holding a plastic cup with a picture of a bear in a high-chair holding a cup with a picture of a bear.....

I always used to wonder how many bears there were and how big the littlest one was. Now, when we are using recursion in our macros we want to be sure we know that there is a last little bear -- a last call -- a way of leaving the lowest level of recursion. Otherwise (since each recursive call uses up some memory/stack space) we will soon run out of room to store the return information and BOOM goes the assembly.

My macro uses the principle of 'divide and conquer' to allocate a chosen number of bytes all of which hold the value we want them to have. We might use this macro with a table of 128 bytes. All non-alphabetic characters (codes $0 to $40, plus assorted others) will have the value $FF in their corresponding bytes in the table. All alphabetics will have a number indicating their relative frequency in English text. We could set up the table with lines and lines of hex strings for all the non-alphabetics. Or we could let the program filter out the alphabetics and use a shorter table. But for the sake of the example let us assume we need the program to be as fast as possible and memory space is no object.
Here is the macro definition I came up with:

```
.MA DB          Macro name is "DB"
.DO ]1<2        If only one left,
.DA ]2             generate a data byte
.ELSE           If more than one left,
>DB ]1/2,]2        call DB for half of them,
>DB ]1+1/2,]2      and call DB again for the other half
.FIN
.EM
```

Here is the table I talked about:

```
>DB $41,#$FF    65 bytes filled with $FF
.HS 00000000    upper case (fill in your own frequencies)
......
>DB 5,#$FF      5 bytes filled with $FF
.HS 00000000    lower case
......
>DB 5,#$FF      5 bytes filled with $FF
```

Here is a sample program to use such a table:

```
LDA CHARACTER.BYTE     get character
TAY
LDA TABLE,Y            get frequency
......                 and then you have to use it
```

The macro calls itself to set up half of the area desired and then calls itself again to set up the other half. The adding of one to the second call makes sure that both odd and even values for the first parameter will work. If the call only needs one byte to be set up then a .DA is used to take care of it. That provides the end of the little bears -- when the first parameter is one.

When I needed a macro like this my first idea was to have each recursive call take care of one byte and then call itself to take care of the rest. If the macro was called with zero repetitions then nothing would be done except end the macro. The problem with that method is the amount of stack space used as the recursion goes to very deep levels. The method used in the example will only recurse, for example, 8 levels to generate 127 bytes of data.

By the way, notice that you must put the pound sign (#) on the second parameter if you want to generate single bytes. Leaving it off will generate two-byte values of data. I chose that method to make the macro more flexible. You might want to put the pound sign (#) inside the macro to make it safer in case you always want to generate single bytes of data. Also, you can use calculated values like #'F+$80 to generate tables of some character value.

The assembly of recursive macros produces quite a few extra lines in the listing, so after checking it out you will probably want to turn off the listing of the macro expansion with ".LIST MOFF". Here is a sample listing with the macro expansion listing on:

```
1000          .MA DB
1010          .DO ]1<2
1020          .DA ]2
1030          .ELSE
1040          >DB ]1/2,]2
1050          >DB ]1+1/2,]2
1060          .FIN
1070          .EM
1080
```

```
0800-          1090          >DB 3,#0
               0000>          .DO 3<2
               0000>          .ELSE
0800-          0000>          >DB 3/2,#0
               0000>>          .DO 3/2<2
0800- 00       0000>>          .DA #0
               0000>>          .ELSE
               0000>>          .FIN
0801-          0000>          >DB 3+1/2,#0
               0000>>          .DO 3+1/2<2
               0000>>          .ELSE
0801-          0000>>          >DB 3+1/2/2,#0
               0000>>>          .DO 3+1/2/2<2
0801- 00       0000>>>          .DA #0
               0000>>>          .ELSE
               0000>>>          .FIN
0802-          0000>>          >DB 3+1/2+1/2,#0
               0000>>>          .DO 3+1/2+1/2<2
0802- 00       0000>>>          .DA #0
               0000>>>          .ELSE
               0000>>>          .FIN
               0000>>          .FIN
               0000>          .FIN
```

INTRODUCING AED][

A POWERFUL AID FOR APPLESOFT PROGRAM DEVELOPMENT

- A full function line editor supports character insertion, single and multiple character deletion, line truncation, instant cursor positioning to a specific character, the beginning, or end of any APPLESOFT line.
- All editing functions are in effect while entering new APPLESOFT statements as well as revising existing lines of your Basic programs.
- Search your APPLESOFT program for a particular string and immediately begin editing the line with the cursor positioned on the matching string.
- Global search and replace one string with another in all or selective lines.
- Wildcard characters supported in searching: /@TAB/ matches HTAB, VTAB, etc.
- User defined ESCAPE MACROS: Insert commonly used Applesoft words and phrases into the line being edited by pressing ESC and a single key.
- Automatic line numbering with user specified increment.
- Sophisticated LIST control allows page-at-a-time, line-at-a-time, fast and slow speed scrolling of APPLESOFT lines. Say goodbye to CTRL-S and CTRL-C. Single key repeat of last LIST command eliminates retyping line number range.
- AED remains resident while RUNning your APPLESOFT program. Single key-stroke exit from AED; reentry to AED via & (ampersand) or pressing RESET.
- A single key command displays the names and values of each APPLESOFT simple variable (very useful in debugging).
- DOS command menu provides CATALOG (D1 or D2), LOAD, SAVE, LOCK, UNLOCK, or DELETE of the current program via one or two keystrokes.
- Direct keyboard entry of [,], \, and _ characters.
- Single key HELP screen and ESC MACRO menu even while editing a line.
- Careful attention to human factors: blinking underscore cursor, soft bell tone for errors, visual mode indicator, and minimal keystroke commands.
- AED is 100% Assembly Language and uses approximately 5K RAM. Apple with 48K RAM, Applesoft ROM, and DOS 3.3 are required.

AED][DISKETTE & USER GUIDE:.....$40.00

DEALER INQUIRIES INVITED

LINN SOFTWARE
3199 HAMMOCK CREEK
LITHONIA, GEORGIA 30058
(404) 483-7637

RWTS CALLER .....by Bill Morgan

Here is a routine to directly call RWTS (Read/Write Track/Sector), the subroutine in DOS that actually reads from or writes to the disk. Many programs use a routine like this to handle disk I/O, without all the time-consuming overhead of the DOS file manager.

All you need to do to use RWTS directly is to place certain information into an Input/Output control Block (IOB), and tell RWTS where the IOB is. Following is an explanation of the IOB (the addresses are those of DOS's own IOB; you can use it yourself, or build your own wherever is convenient):

Address	Description
$B7E8	Table type, always $01
$B7E9	Slot number times 16, usually $60
$B7EA	Drive number, $01 or $02
$B7EB	Volume number expected, $00 matches anything
$B7EC	Track number, $00 through $22
$B7ED	Sector number, $00 through $0F
$B7EE-F	Address of Device Characteristics Table, $B7FB for DOS's own DCT
$B7F0-1	Address of buffer, wherever you want
$B7F2	Not used
$B7F3	Byte count if partial sector, $00 normally
$B7F4	Command $00 = SEEK $01 = READ $02 = WRITE $04 = FORMAT
$B7F5	Error Code $00 = No errors $08 = Error in initialization $10 = Write protect error $20 = Volume mismatch $40 = Drive error
$B7F6	Last volume number
$B7F7	Last slot number times 16
$B7F8	Last drive number

The Device Characteristics Table (whose address is at $B7EE,EF) is a four-byte block containing information about the disk drive. For a standard Apple Disk II this block always contains $00 01 EF D8.

For our purposes, the most important items in the IOB are track, sector, buffer address, and command. By manipulating these, you can read any sector of the disk into any area of memory. All you need to do is set up the IOB, load the A- and Y-registers with the address of the IOB, and JSR $3D9. And if you decide to use the file manager's IOB, you can even set up the A- and Y-registers by a simple JSR $3E3.

RWTS will read the track and sector you choose into your 256-byte buffer. If there was a disk error, RWTS will return with the carry bit set and an error code stored in the IOB. It is then up to the user's program to decide what to do about the error. If there was no error, carry will be clear.

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

QUICKTRACE *$50*

Is a trademark of Anthro-Digital, Inc.

Written by John Rogers

See these programs at participating Computerland and other fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

This month we'll set up a short program using the RWTS Caller to read an entire track of the disk into 16 consecutive pages of memory. DOS stores information on a track starting at sector $0F and working back to sector $00, so we must read a sector into the buffer, decrement the sector in the IOB, and increment the buffer pointer.

RWTS.CALLER:

Lines 1680-1850 set up the IOB, transferring values from the program variables.

Lines 1870-1890 load the address of the IOB and call RWTS.

Lines 1900-1910 are necessary to avoid confusing the system monitor. (RWTS and the monitor both use location $48.)

Lines 1960-2030 ring a warning if a disk error occurred, and display the error code.

TRACK READ:

Lines 1260-1350 initialize the variables and call input routines.

Lines 1370-1440 read the sectors from $0F through $00 into the buffer. Line 1410 will end the program if an error occurred.

Line 1460 will become a display routine. (Or, whatever processing you want to do on the buffer.)

Lines 1500-1650 will become input routines; right now they just set the track, buffer and command variables.

CAUTIONS:

1) These routines have very little error-checking. It is very easy to make a trivial error and lose information from a diskette. Always test an RWTS-calling program on a diskette you don't care about.

2) If you store information on a blank area of a diskette using these techniques, DOS doesn't know you have taken some space. Unless you modify the VTOC to show that the sectors are used, DOS can overwrite your data. (What's a VTOC?, you say. Volume Table of Contents. We'll go into that another time.)

There is more about RWTS on pages 94-98 of the Apple DOS Manual, and a goldmine of information in Beneath Apple DOS, by Don Worth and Pieter Lechner. (Quality Software, 1981.)

```
                    1000 *SAVE TRACK READ
                    1010 *-------------------------------
0000-               1020 SLOT     .EQ $60      $60 OR $70
0001-               1030 DRIVE    .EQ $01      1 OR 2
0002-               1040 VOLUME   .EQ $02      0 = DON'T CARE
0003-               1050 TRACK    .EQ $03      $00 TO $22
0004-               1060 SECTOR   .EQ $04      $00 TO $0F
0005-               1070 BUFFER   .EQ $05,06
0007-               1080 COMMAND  .EQ $07      1 = READ, 2 = WRITE
0048-               1090 PREG     .EQ $48
                    1100 *
03D9-               1110 RWTS     .EQ $3D9
                    1120 *
B7E8-               1130 IOB          .EQ $B7E8    DOS'S OWN IOB
B7E9-               1140 IOB.SLOT     .EQ $B7E9
B7EA-               1150 IOB.DRIVE    .EQ $B7EA
B7EB-               1160 IOB.VOLUME   .EQ $B7EB
B7EC-               1170 IOB.TRACK    .EQ $B7EC
B7ED-               1180 IOB.SECTOR   .EQ $B7ED
B7F0-               1190 IOB.BUFFER   .EQ $B7F0,F1
B7F4-               1200 IOB.COMMAND  .EQ $B7F4
B7F5-               1210 IOB.ERROR    .EQ $B7F5
                    1220 *
FDDA-               1230 PRBYTE .EQ $FDDA
FDED-               1240 COUT   .EQ $FDED
                    1250 *-------------------------------
                    1260 SETUP
0800- A9 60         1270         LDA #$60
0802- 85 00         1280         STA SLOT      SLOT 6
0804- A9 01         1290         LDA #$01
0806- 85 01         1300         STA DRIVE     DRIVE 1
0808- A9 00         1310         LDA #$00
080A- 85 02         1320         STA VOLUME    ANY VOLUME
080C- 20 25 08      1330         JSR GET.TRACK
080F- 20 2A 08      1340         JSR GET.BUFFER
0812- 20 33 08      1350         JSR GET.COMMAND
                    1360 *-------------------------------
                    1370 READ.TRACK
0815- A9 0F         1380         LDA #$0F      START AT SECTOR $0F
0817- 85 04         1390         STA SECTOR
0819- 20 38 08      1400 .1      JSR RWTS.CALLER   READ ONE SECTOR
081C- B0 06         1410         BCS EXIT      EXIT IF ERROR
081E- E6 06         1420         INC BUFFER+1  NEXT BUFFER PAGE
0820- C6 04         1430         DEC SECTOR    NEXT SECTOR
0822- 10 F5         1440         BPL .1        NOT DONE, READ NEXT SECTOR
                    1450 *-------------------------------
                    1460 DISPLAY
                    1470 *-------------------------------
0824- 60            1480 EXIT     RTS
                    1490 *-------------------------------
                    1500 GET.TRACK
0825- A9 11         1510         LDA #$11
0827- 85 03         1520         STA TRACK     TRACK $11 (DIRECTORY)
0829- 60            1530         RTS
                    1540 *-------------------------------
                    1550 GET.BUFFER
082A- A9 00         1560         LDA #0
082C- 85 05         1570         STA BUFFER    BUFFER AT $4000
082E- A9 40         1580         LDA #$40
0830- 85 06         1590         STA BUFFER+1
0832- 60            1600         RTS
                    1610 *-------------------------------
                    1620 GET.COMMAND
0833- A9 01         1630         LDA #1
0835- 85 07         1640         STA COMMAND   READ
0837- 60            1650         RTS
                    1660 *-------------------------------
                    1670 RWTS.CALLER
0838- A5 00         1680         LDA SLOT      TRANSFER
083A- 8D E9 B7      1690         STA IOB.SLOT  VALUES
083D- A5 01         1700         LDA DRIVE      INTO
083F- 8D EA B7      1710         STA IOB.DRIVE   IOB
0842- A5 02         1720         LDA VOLUME
0844- 8D EB B7      1730         STA IOB.VOLUME
0847- A5 03         1740         LDA TRACK
0849- 8D EC B7      1750         STA IOB.TRACK
084C- A5 04         1760         LDA SECTOR
084E- 8D ED B7      1770         STA IOB.SECTOR
```

```
0851- A5 07     1780         LDA COMMAND
0853- 8D F4 B7  1790         STA IOB.COMMAND
0856- A5 05     1800         LDA BUFFER
0858- 8D F0 B7  1810         STA IOB.BUFFER
085B- A5 06     1820         LDA BUFFER+1
085D- 8D F1 B7  1830         STA IOB.BUFFER+1
0860- A9 00     1840         LDA #$00
0862- 8D F5 B7  1850         STA IOB.ERROR
                1860 *--------------------------------
0865- A0 E8     1870         LDY #IOB      LOAD IOB
0867- A9 B7     1880         LDA /IOB      ADDRESS
0869- 20 D9 03  1890         JSR RWTS      CALL RWTS
086C- A9 00     1900         LDA #$00
086E- 85 48     1910         STA PREG      SOOTHE MONITOR
0870- B0 01     1920         BCS ERROR.HANDLER
0872- 60        1930         RTS
                1940 *--------------------------------
                1950 ERROR.HANDLER
0873- A9 87     1960         LDA #$87      BELL
0875- 20 ED FD  1970         JSR COUT      RING
0878- 20 ED FD  1980         JSR COUT         ING
087B- 20 ED FD  1990         JSR COUT           ING
087E- AD F5 B7  2000         LDA IOB.ERROR
0881- 20 DA FD  2010         JSR PRBYTE    DISPLAY ERROR CODE
0884- 38        2020         SEC           EXIT WITH CARRY SET
0885- 60        2030         RTS
```

Reading the Game Buttons.....Jim Kassel

Recently I was asked to come up with a maching language subroutine that involved using the Apple game buttons. Fortunately for me at the time, I forgot that my paddles were not plugged in. I was in for a rude awakening because when I tested the program, it said that all of the buttons were constantly being pushed!

I needed some additional programming to check whether the buttons were even plugged in. The problem occurs because the Apple button logic returns the same value for a pushed button as for a missing button. To get technical: in TTL logic, when an input pin of an IC chip is left unconnected, the chip thinks the pin is at a logic "1". The Apple buttons supply a logic "1" when they are plugged in and pushed. Hence, the hardware cannot tell the difference between a plugged-in-pushed-down button and a missing button.

What the hardware does know for sure is when a button is plugged in and not pushed. This is the only case in which a logic "0" is developed. I had to use this knowledge to write a program which could tell what a logic "1" really means. Since an installed unpushed button does unambiguously announce its presence by a "0" in bit 7 of the input byte, I could make a mask indicating which buttons appear to be installed.

I started by writing the GET.BUTTON.STATUS subroutine. It reads each of the three buttons, and packs the three bit-7's into one byte. The way I wrote it, bit 7 of the returned byte represents button 1, bit 6 is button 2, and bit 5 is button 3. If a button is installed and not pushed, the corresponding bit will be "1"; otherwise it will be "0".

Look at the listing, lines 1250-1350, and I'll describe how GET.BUTTON.STATUS works. I used an indexed loop, where X goes from 2 to 0, step -1, addressing all three of the buttons. Only bit 7 of a button byte is significant. I invert this bit (line 1280), and shift it into the carry status bit (line 1290). Then line 1300 rolls the bit into GB.PUSH. After all three have been read and rolled, I pick up GB.PUSH and zero out the lower five bits at line 1340.

Now lets look at the other three subroutines.
GAME.BUTTON.INITIALIZE simply clears out GB.STAT. We have to start with GB.STAT = 0, so that as we discover each installed button we can set its bit. Call this subroutine once at the beginning of your overall run.

GAME.BUTTON.INSTALLED reads the current button status; remember that a "1" here indicates an installed but unpushed button. So line 1140 merges all "1" bits into GB.STAT. You need to call this subroutine several times, at time intervals of several seconds at least, to be sure that every installed button is noticed at least once. (The first few times you call it, you might be pushing down an installed button; then finally you let go, so this subroutine sees the button.)

GAME.BUTTON.PUSHED reads the current button status, and with a little boolean logic comes out with the final result: a "1" indicating an installed and pushed button, and a "0" meaning either a missing button or an unpushed button. The result is in the A-register, and also in GB.PUSH.

Here is a truth table of the logic involved:

GB.STAT	CURRENT READING	EOR STAT	AND STAT
0 (no button)	0 (none or pushed)	0	0
0 (no button)	1 (unpushed)	1	0
1 (button)	0 (none or pushed)	1	1
1 (button)	1 (unpushed)	0	0

There are other possible complications in reading buttons, which I have not handled here. You might want to "debounce" the buttons, so that you don't get false indications of multiple pushes when the button begins to make or break contact. And, you might want to guarantee that any action which happens from a pushed button only happens once per push.

Lines 1400-1910 demonstrate the usage of the subroutines. After clearing the screen, the buttons are continuously monitored until you press any key on the keyboard. The status of each button will be displayed on the screen: button 1 on the to line, button 2 on the second line. and button 3 on the third line. If you hold a button pushed and then start the program, it will say "not installed" until you release the button; from then on it will track the button properly.

If you have the shift key mod installed in button 3, it will say "not installed" until you press the shift key; from then on it will say "pushed" when you are not pushing, and "not pushed" when you are. This is because the sense of the shift key is the opposite of the normal game paddle buttons.

```
                    1000 *--------------------------------
                    1010 *      GAME BUTTON SUBROUTINES
                    1020 *--------------------------------
C061-               1030 GAME.BUTTON .EQ $C061  BASE ADDRESS
                    1040 *--------------------------------
                    1050         .OR $800
                    1060 *--------------------------------
                    1070 GAME.BUTTON.INITIALIZE
0800- A9 00         1080         LDA #0
0802- 8D 31 08      1090         STA GB.STAT
0805- 60            1100         RTS
                    1110 *--------------------------------
                    1120 GAME.BUTTON.INSTALLED
0806- 20 1D 08      1130         JSR GET.BUTTON.STATUS
0809- 0D 31 08      1140         ORA GB.STAT  SET BITS OF ANY BUTTONS
080C- 8D 31 08      1150         STA GB.STAT  PLUGGED IN AND NOT PUSHED
080F- 60            1160         RTS
                    1170 *--------------------------------
                    1180 GAME.BUTTON.PUSHED
0810- 20 1D 08      1190         JSR GET.BUTTON.STATUS
0813- 4D 31 08      1200         EOR GB.STAT  MASK OUT BUTTONS WHICH
0816- 2D 31 08      1210         AND GB.STAT    ARE NOT PLUGGED IN
0819- 8D 32 08      1220         STA GB.PUSH
081C- 60            1230         RTS
```

```
                        1240 *--------------------------------
                        1250 GET.BUTTON.STATUS
081D- A2 02             1260         LDX #2
081F- BD 61 C0          1270 .1      LDA GAME.BUTTON,X
0822- 49 80             1280         EOR #$80       INVERT SENSE
0824- 0A                1290         ASL            INTO CARRY BIT
0825- 6E 32 08          1300 .2      ROR GB.PUSH
0828- CA                1310         DEX            NEXT BUTTON
0829- 10 F4             1320         BPL .1
082B- AD 32 08          1330         LDA GB.PUSH
082E- 29 E0             1340         AND #$E0       CLEAR EXTRANEOUS BITS
0830- 60                1350         RTS
                        1360 *--------------------------------
0831-                   1370 GB.STAT .BS 1
0832-                   1380 GB.PUSH .BS 1
                        1390 *--------------------------------
0025-                   1400 MON.CV      .EQ $25
FC58-                   1410 MON.HOME    .EQ $FC58
FC22-                   1420 MON.VTAB    .EQ $FC22
FC9C-                   1430 MON.CLREOL  .EQ $FC9C
FDED-                   1440 MON.COUT    .EQ $FDED
0000-                   1450 TEST.MASK   .EQ $00
                        1460 *--------------------------------
0833- 20 58 FC          1470 TEST    JSR MON.HOME
0836- 20 00 08          1480         JSR GAME.BUTTON.INITIALIZE
0839- A9 00             1490 .1      LDA #0
083B- 85 25             1500         STA MON.CV
083D- 20 22 FC          1510         JSR MON.VTAB
0840- 20 06 08          1520         JSR GAME.BUTTON.INSTALLED
0843- 20 10 08          1530         JSR GAME.BUTTON.PUSHED
0846- A9 84             1540         LDA #$84
0848- 85 00             1550         STA TEST.MASK
084A- A5 00             1560 .2      LDA TEST.MASK
084C- 2D 32 08          1570         AND GB.PUSH
084F- D0 0A             1580         BNE .3         PUSHED
0851- A5 00             1590         LDA TEST.MASK
0853- 2D 31 08          1600         AND GB.STAT
0856- D0 06             1610         BNE .4         NOT PUSHED
0858- A0 10             1620         LDY #QTGONE-QTS   NOT INSTALLED
085A- 2C                1630         .HS 2C
085B- A0 00             1640 .3      LDY #QTPUSHED-QTS
085D- 2C                1650         .HS 2C
085E- A0 06             1660 .4      LDY #QTNOTPSH-QTS
0860- 20 72 08          1670         JSR MSGOUT
0863- 46 00             1680         LSR TEST.MASK
0865- 90 E3             1690         BCC .2
0867- AD 00 C0          1700         LDA $C000
086A- 10 CD             1710         BPL .1
086C- 8D 10 C0          1720         STA $C010
086F- 60                1730         RTS
0870- B0 C7             1740         BCS .1         ...ALWAYS
                        1750 *--------------------------------
0872- B9 87 08          1760 MSGOUT LDA QTS,Y
0875- 48                1770         PHA
0876- 09 80             1780         ORA #$80
0878- 20 ED FD          1790         JSR MON.COUT
087B- C8                1800         INY
087C- 68                1810         PLA
087D- 10 F3             1820         BPL MSGOUT
087F- 20 9C FC          1830         JSR MON.CLREOL
0882- A9 8D             1840         LDA #$8D
0884- 4C ED FD          1850         JMP MON.COUT
                        1860 *--------------------------------
                        1870 QTS
0887- 50 55 53
088A- 48 45 C4          1880 QTPUSHED    .AT /PUSHED/
088D- 4E 4F 54
0890- 20 50 55
0893- 53 48 45
0896- C4                1890 QTNOTPSH    .AT /NOT PUSHED/
0897- 4E 4F 54
089A- 20 49 4E
089D- 53 54 41
08A0- 4C 4C 45
08A3- C4                1900 QTGONE      .AT /NOT INSTALLED/
                        1910 *--------------------------------
```

Macro Branch Library.....R. F. O'Brien

When I received my copy of the S-C Macro Assembler, my first task was to make up a set of branch macro definitions to use in all my programs. This set will finally eliminate the need to check usage of BCC, BCS, etc., and generally make the programs more readable.

There are six branch-on-tests: >BLT, >BLE, >BGE, >BGT, >BEQ, AND >BNE. All of these would normally be used with two parameters, e.g. >BGT P1,P2...reads: if contents of accumulator is greater than P1 then branch to P2. The first four of these can be gainfully used with only one parameter, after a comparison. Sample program:

```
LDA #$8D
CMP #$8E
>BLT THERE ... results in a branch to THERE
```

While >BEQ and >BNE are defined so as to work with only one parameter, there is no reason to so use them - it is easier to just use BEQ and BNE.

The macro >BRA (BRanch Always) is used with one parameter - any others are ignored. >BRA LABEL causes a jump to LABEL, up to +- 127 bytes away. To overcome this limitation I decided to put the macro facility to good use to provide for easy branching to any part of a program - this is necessary for writing relocatable code. I settled for a two-paramter code >JMP:

>JMP P1,P2 ... where P1 is the intermediate or final label you wish to branch to and P2 is the label for this instruction.

Instructions such as the foregoing are inserted anywhere you wish in the program (within 127 bytes of each other) to allow for unlimited branching whilst retaining relocatable code. The following is an example of how you might use the >JMP code:

```
1000 A    etc.
          (more code....)
2000      >JMP A,B
          (more code....)
3000      >BRA B
```

When a program designed as the above is run it will simulate an absolute jump to A. The >BRA B will branch to label B, which contains a >BRA A. This sequence of instructions is transparent to the rest of the program, as the first instruction in the >JMP is to skip around the the >BRA within the definition.

This use of macro definitions can easily be extended to the X and Y registers; simply substitute CPX's or CPY's for the CMP's.

Following are some examples of these macros at work:

```
>BLT #3,THERE....if (A) is less than 3 then go THERE
>BGT $40,THIS....if (A) is greater than ($40) then go THIS
>BEQ #'A,THAT....if (A) is equal to $41 then go THAT
```

To use these macros in all your programs, place the command .IN MACRO.BRANCH.LIBRARY at the beginning of your source program.

```
1000 *  MACRO BRANCH LIBRARY
1010 *  BY R.F. O'BRIEN
1020 *--------------------------------
1030 *    >BLT P1 (,P2)   BRANCH IF (A) < P1...TO P2
1040          .MA BLT
1050          .DO ]#>1
1060          CMP ]1
1070          BCC ]2
1080          .ELSE
1090          BCC ]1
1100          .FIN
1110          .EM
1120 *--------------------------------
1130 *    >BLE P1 (,P2)   BRANCH IF (A)<=P1...TO P2
1140          .MA BLE
1150          .DO ]#>1
1160          CMP ]1
1170          BEQ ]2
1180          BCC ]2
1190          .ELSE
1200          BEQ ]1
1210          BCC ]1
1220          .FIN
1230          .EM
1240 *--------------------------------
1250 *    >BGE P1 (,P2)   BRANCH IF (A)>=P1...TO P2
1260          .MA BGE
1270          .DO ]#>1
1280          CMP ]1
1290          BCS ]2
1300          .ELSE
1310          BCS ]1
1320          .FIN
1330          .EM
1340 *--------------------------------
1350 *    >BGT P1 (,P2)   BRANCH IF (A)>P1...TO P2
1360          .MA BGT
1370          .DO ]#>1
1380          CMP ]1
1390          BEQ :1
1400          BCS ]2
1410 :1
```

```
1420         .ELSE
1430         BEQ :1
1440         BCS ]1
1450 :1
1460         .FIN
1470         .EM
1480 *--------------------------------
1490 *    >BRA P1          BRANCH ALWAYS...TO P1
1500         .MA BRA
1510         CLV
1520         BVC ]1
1530         .EM
1540 *--------------------------------
1550 *    >BEQ P1 (,P2)    BRANCH IF (A)=P1...TO P2
1560         .MA BEQ
1570         .DO ]#>1
1580         CMP ]1
1590         BEQ ]2
1600         .ELSE
1610         BEQ ]1
1620         .FIN
1630         .EM
1640 *--------------------------------
1650 *    >BNE P1 (,P2)    BRANCH IF (A)<>P1...TO P2
1660         .MA BNE
1670         .DO ]#>1
1680         CMP ]1
1690         BNE ]2
1700         .ELSE
1710         BNE ]1
1720         .FIN
1730         .EM
1740 *--------------------------------
1750 *    >JMP P1,P2       BRANCH ALWAYS TO P1 BY
1760 *                     BRANCHING TO P2 (SEE ARTICLE)
1770         .MA JMP
1780         CLV
1790         BVC :1
1800 ]2      CLV
1810         BVC ]1
1820 :1
1830         .EM
```

Good Price on the NEC printers

I can ship you an NEC PC-8023A-C dot matrix printer for only $595. I believe the normal list price is $795, but mail order prices are generally less. I also have the Grappler interface card and cable, configured for the NEC printer, for only $150 (normally $175, I think). And if you want both printer and interface at the same time, the combined price is only $695.

I have two of these printers, and like them better than my Epson MX-80. Why? Faster: 100cps instead of 80cps. Fully equipped: standard features include graphics, tractor feed, and friction feed. Handier: the friction feed is just like a typewriter, platen and all; and option switches, should you wish to change them, are accessible without removing any screws. I run one of them with an Epson parallel interface, and the other with the Grappler.

If you would rather have a Spinwriter (that is what this newsletter is printed on), call me for prices.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; $18 per year sent First Class Mail in USA, Canada, and Mexico; $28 per year sent Air Mail to other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).